

Primer

In the 1960s, a group of engineers and mathematicians working at AT&T Bell Labs invented a set of techniques known as digital-signal-processing (DSP). The problem they were working on was the recovery of information from signals that were distorted, noisy, incomplete, and ambiguous—data transmission over long-distance phone lines and satellite links. The algorithms they invented are used today in many fields, including stereos, medical imaging, high-speed communications, and servo controllers.

Unfortunately, most presentations of the mathematics of this field are rather complicated and unintuitive. The purpose of this two-part series is to give you a good understanding of these techniques and to help you develop an intuition for when and how to use them. This presentation does not explain the algorithms in their full mathematical gore and glory. To understand this month's discussion of digital filtering, you need to understand vectors and Fourier transforms, which were covered in last month's article ("A DSP Primer," by Mark Lawrence, *Embedded Systems Programming*, Sept. 1991, pp. 50-58). The digital-filtering techniques we'll be looking at will enable you to remove unwanted parts of a signal and enhance and clarify the good parts.

USING THE FOURIER TRANSFORM

Everyone is familiar with stereo tone controls—one knob makes the treble (high frequencies) louder and softer, and another knob makes the bass (low frequencies) louder and softer. Many stereos have a graphic equalizer—a set of knobs or slides that control specific frequency bands. You

The techniques we'll be looking at will enable you to remove unwanted parts of a signal and enhance the good parts.

can use a Fourier transform to make a digital graphic equalizer.

After using a Fourier transform on a time series, the result is a set of amplitudes (and phases) at various frequencies. If you want to boost the bass and cut the treble, you just select the appropriate frequencies and multiply their amplitudes by the desired factor. If you're working on music, 3dB is a factor of two, so multiplying an amplitude by 16 will boost that frequency by 12dB, and dividing an amplitude by four will cut that frequency by 6dB. While you're at it, you can take this opportunity to mess around with the phase. When you're done, just run the data through the Fourier transform again and out comes your improved time series. More generally, you would draw a filtering function and multiply the amplitudes by the corresponding value in the filtering function.

Figure 1 shows several common filters. A low-pass filter allows low frequencies through but stops high frequencies. High frequencies often con-

Irina Rosovskaya

Son of DSP Primer

tain noise such as tape hiss. A high-pass filter allows high frequencies through but stops low frequencies. Low frequencies also often contain noise such as subsonic rumble. Band-pass filters are used when only a narrow frequency band contains useful information. Televisions and radios use band-pass filters tuned to the channels. Band-stop filters stop a certain narrow frequency band. Most places have a lot of 60Hz and 120Hz noise due to power lines and fluorescent lights. Frequency-shaping filters also accomplish certain tasks. Phonograph records are recorded with a particular RIAA encoding, and must be decoded

through a frequency-shaping filter.

Using an fast Fourier transform (FFT), you can perform any combination of these filtering functions on your data, then return it to its normal form. With an FFT, you can accomplish anything digitally that can be done with analog filters (capacitors and inductors). You can also do things that cannot be accomplished by these means.

THE EFFECTS OF FILTERING

It all seems so easy—there must be a catch. Well, in fact, several exist. Normally, these catches make no difference, but you should know a few

Briefly, a perfect filter violates causality. To make a perfect filter, you need to be able to see into the future—accurately.

things about them.

First, none of the filters shown in Figure 1 can actually be made. You can only make approximations. A realizable low-pass filter has a pass band with a certain height, a transition region with a certain width, and a stop band with a certain height, as shown in Figure 2.

If you're using a digital computer and a Fourier transform, problems are created by the finite precision of the machine and the finite number of data points in the FFT. Due to the finite precision, you cannot specify exact results

Figure 1
Several common types of filters.

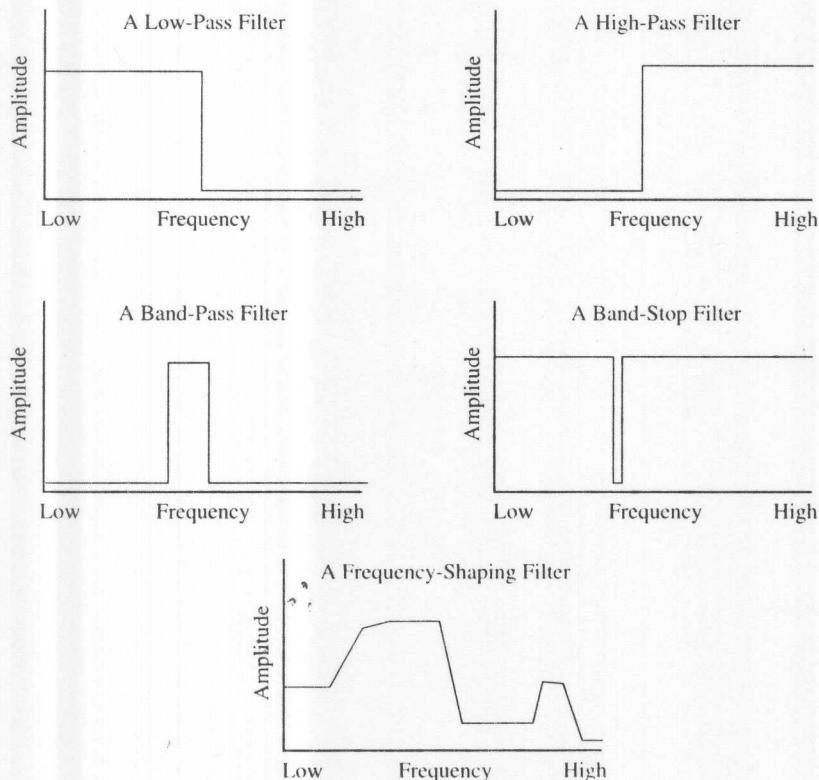
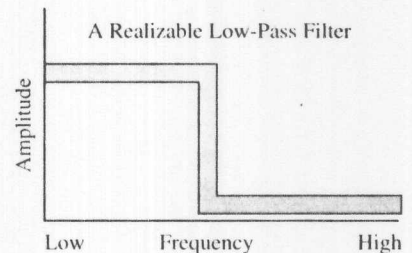


Figure 2
Pass band, transition region, and stop band for a filter.

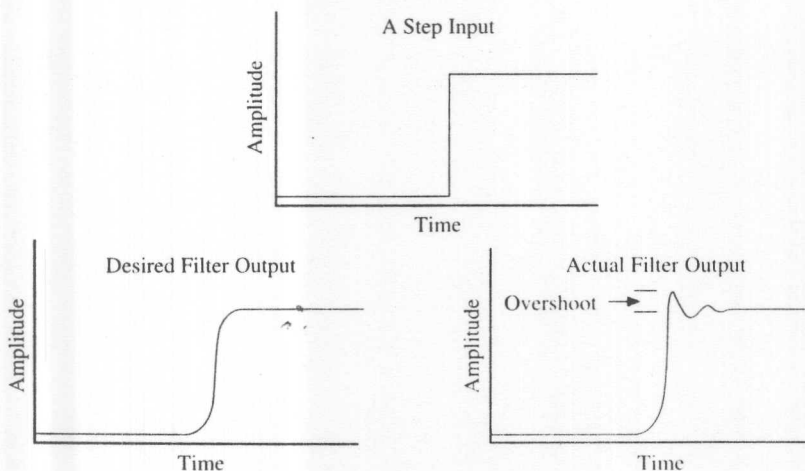


from the arithmetic operations. The filter specification may be made of ones and zeros, but the FFT is made of complex exponentials. The transition region is at least the difference in the two adjacent frequencies of the FFT.

Perhaps some exact process of arithmetic or some different type of Fourier-class transform that represents all frequencies can help us around this problem. For example, what if the data were represented functionally instead of numerically, and the Fourier transform was done with analytic integrals? It's still impossible, as I'll show you a bit later. The story of science and math in the 20th century is the discovery of limits such as the speed of light, the uncertainty principle, and Godel's incompleteness theorem. Briefly, a perfect filter violates causality. To make a perfect filter, you need to be able to see into the future—accurately. (Glendower: "I can summon spirits from the vasty deep!" Falstaff: "Aye, and so can I, or any other man. But do they come when you call?"")

Figure 3

Damped and undamped filter responses.



Usually, the precision and memory of digital computers allows us to create filters very well. CDs are recorded with 16-bit integers at 44kHz and have a remarkably good reputation for sound quality.

Another, more practical limitation has to do with stability. Suppose we're implementing a low-pass filter. Low-pass filters remove high frequencies. High frequencies make steep slopes and sharp corners. Let's look at the step response of our low-pass filter. The input waveform will be a step: the input is zero for a certain period of time, then one after that. We want to see an output that is much like the input, but with rounded corners. What we're more likely to see is the waveform shown in Figure 3. Engineers call these waveforms damped and undamped response.

Imagine hitting a cymbal—the resulting ring is an undamped oscillation. Now, imagine wrapping a towel around the cymbal and holding it with one hand. Now when you hit it, it goes "thud." This effect is called a damped

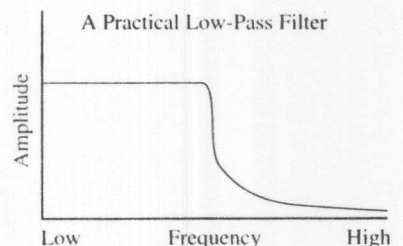
oscillation. Often, undamped response is very bad—if the step input is a command to your hard disk to move to the next track and the response is undamped, you'll have to wait a long time before you can read your data.

The amount of overshoot has to do with the steepness of your filter. If your filter is all ones for a while, then all zeros, you will have overshoot. A mathematician named Gibbs discovered that if your filter is zeros and ones, now matter how hard you try to use more precision, more data points, and a better FFT, the overshoot will always be at least 7%. This effect is called the Gibbs phenomenon. If you cannot tolerate overshoot in your system, you must use a filter without sharp corners or vertical lines, as shown in Figure 4.

The shape you choose for your filter is called the window. Designing filters by shape considerations is called the windowing method of filter design. Many shapes have been mathematically described and investigated: rectangles, triangles, Kaiser windows, Hamming windows, Hanning windows (no one can ever keep these two straight), Chebyshev windows, and so on. They're all a minor variant on the shape shown in Figure 4. The distinctions between them are not very important unless you

Figure 4

A filter without sharp corners or vertical lines.



Son of DSP Primer

work for AT&T, where a 1% data-throughput difference makes a 1% difference in income.

FILTERING WITHOUT FFTS

Do you really need to use FFTs to solve your data-filtering problem? Maybe not. In the filtering methods we've looked at so far, we take our data, perform an FFT, process the resulting frequency-domain data with our chosen filter function, and perform another FFT to get back to the time domain. When we're in the frequency domain, we multiply our data by our chosen filter function. Does a time-domain operation that does the same thing as multiplication in the frequency domain exist? The answer is yes. The operation is called convolution.

First, the math. If your data is $D(t)$, your filter function is $F(\omega)$, and the Fourier transform of your filter function is $F(t)$, the filter operation is:

$$D'(t) = \text{Fourier}(F(\omega) \text{Fourier}(D(t)))$$

$$D'(t) = \text{Convolution}(\text{Fourier}(F(\omega)), D(t))$$

$$D'(t) = \int_{-\infty}^{\infty} D(x) F(t-x) dx$$

CONVOLUTION

Convolution is much easier to see graphically. Suppose we want to convolve functions A and B, as shown in Figure 5. First, we mirror B using the Y axis. Then, we start moving B to the right, keeping track of how much area the figures have in common. More precisely, we multiply A and B point by point. The final result, the convolution, is this area as a function of the displacement of B. If A is a signal and B is the Fourier transform of a filtering function, this process accomplishes the filtering.

Designing filters by shape considerations is called the windowing method of filter design.

Convolution filters aren't particularly difficult to create. Filter functions all look more or less like Figure 6. For different filter types, the wiggles differ in height, width, and depth.

A perfect filter has nonzero values for $t > \text{now}$; that is, to compute the result, you need to know the future. The

convolution filters I've just described follow this function:

$$\text{output}(t) = \sum \text{filter}(i) \text{input}(t-i)$$

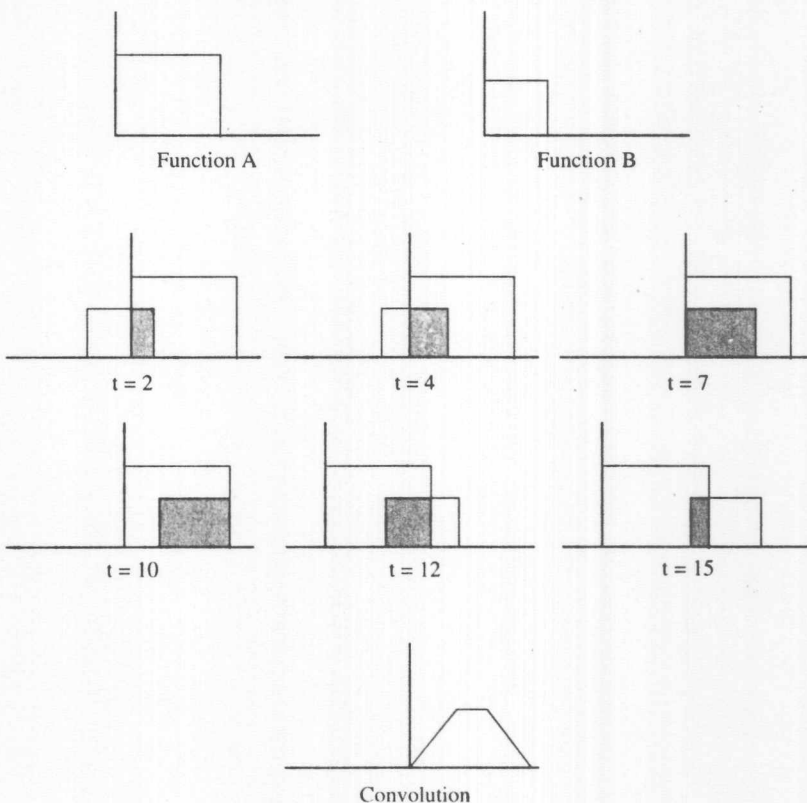
A more general filter would take the previous output into account:

$$\text{output}(t) = \sum \text{filter}_{\text{input}}(i) \text{input}(t-i) + \sum \text{filter}_{\text{output}}(j) \text{output}(t-j)$$

This filter function includes feedback. The previous outputs from the filter are fed back into the filter function to help determine the next output.

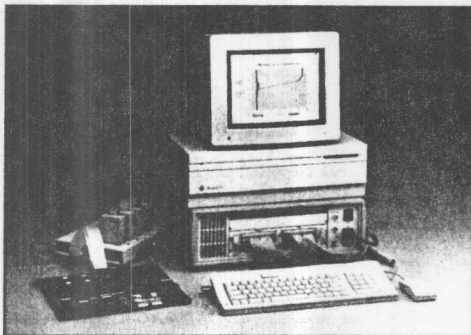
The first filter type depends only on the previous inputs and is called a finite impulse-response (FIR) filter. A digital impulse is a data set that has a value of one at $t=0$ and a value of zero for all other t s. If you put an impulse through an FIR filter, the filter can have nonzero outputs at datapoints from $t=0$ to $t=n$, where n is the number of data points in the filter function. The im-

Figure 5
Graphical convolution.



Biomation's New VVM-1

Hardware For Software Analysis



Reduce debugging, tuning, and testing times by 90% with Biomation's new concept in software analysis, the Variable Value Monitor.

Save time monitoring variable values—They are the landmarks of where the code is going, and where it has been. VVM-1 continuously captures the values at 1024 memory locations.

Save time with flexible graphics—A picture is worth a thousand words. See what you want, displayed the way you want to.

Save time with real-time, non-intrusive capture—Watch behavior at its natural rate of speed. VVM-1 continuously displays 1000 samples per second.

Debug, Tune, and Test

Debugging: Watch process switching behavior before the sector address in the disk driver gets clobbered.

Performance Analysis: Measure the time between sending a network packet and receiving its corresponding acknowledgement.

Testing: Graph motor controller joystick and motor speed inputs, and throttle outputs, verifying acceleration and deceleration ramp behavior is met in different load conditions.

TIME IS MONEY—One VVM-1 customer said "I did a day's worth of debugging in 5 minutes!" Don't spend another day without a VVM-1 which, including the data collection unit, probes, LabVIEW™, and a Macintosh® LC with a 40MB drive and 13-inch color monitor, sells for \$19,950.

Biomation Corporation, 19050 Pruneridge Avenue, Cupertino, CA 95014, 800-538-9320, (408) 988-1647 FAX.

LabVIEW is a trademark of National Instruments Corp. Macintosh is a registered trademark of Apple Computer, Inc.

CIRCLE #238 ON READER SERVICE CARD

Butterworth filters are maximally flat. This filter has the flattest frequency response in the pass band.

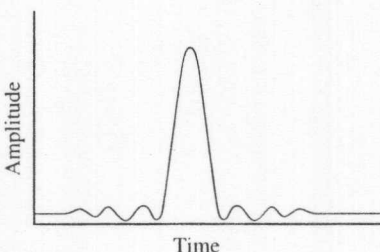
pulse response is finite in duration.

The impulse response of an FIR filter exactly equals the filter's coefficients. You can design an FIR filter by deciding what filter function you need and calculating the impulse response of the filter. This method is commonly used to design FIR filters.

The second filter type, which is based on the previous inputs and outputs, is called an infinite impulse-response (IIR) filter. Since previous outputs are fed back into the filter's input, this filter will respond to its own previous outputs. After putting an impulse through an IIR filter, the filter will have nonzero outputs forever. The impulse response is infinite in duration.

In either case, to filter the data stream we perform a convolution. For each output point, we take the filter function, as shown in Figure 6, line it up properly with the data stream, multiply corresponding points, and add up the result. The result is one filtered data point, which is the reason the FFT was such a breakthrough. Using an FFT, the

Figure 6
A typical filter.



computation required to perform a filtering function is much lower. However, in some cases, FIR and IIR convolution filters are still used. For example, to use an FFT, you have to save up data points until you have enough for an FFT—perhaps 256 data points. If you have enough computing resource and need your filtered results as quickly as possible after each data point comes in, convolution filters are the way to go.

ROLLING YOUR OWN

To use a filter, you have to decide on the frequency shape. You can, of course, draw what you want in the frequency domain. This method is not usually how a shape is decided upon. Filter theory has been mathematically investigated, and it is now understood that certain filters are optimal for certain tasks. The basic filter types are shown in Figure 7.

Butterworth filters are maximally flat. This filter has the flattest frequency response in the pass band. Butterworth filters have somewhat linear phase response.

Bessel filters have maximally flat phase response. All of the other types of filters have nonlinear phase effects. Bessel filters have almost linear phase response. Linear phase response is the same as a time delay. Nonlinear phase response will skew an information packet out in time. Some information will arrive early and some will arrive late. If flat phase response is critical, this filter is the best one to use.

Chebyshev filters are maximally steep. They have ripples in the pass band. You can choose how tall the ripples are when you design the filter. In return, the filter has a very steep transition region. Chebyshev filters have very nonlinear phase response.

Inverse Chebyshev filters are maximally flat in the pass band (they are mathematically identical to Butterworth filters in the pass band) and have ripples in the stop band. Inverse Chebyshev filters have somewhat linear phase response.

Elliptic filters have ripples in the pass band and the stop band. They are, by far, the steepest filter in the transition region. Elliptic filters are based on elliptic integrals. They are extremely difficult to design and have totally in-

Son of DSP Primer

sane phase response.

Each filter can be used as a low-pass, high-pass, band-pass, or band-stop filter. The high-pass versions look like mirror images of the low-pass versions. The band-pass versions look like a low-pass and a high-pass version stuck together. The band-stop versions look like a high-pass and a low-pass version stuck together.

These filters have specific mathematical forms. We won't go into detail about these forms. Normally, you would design one of these filters by looking up the appropriate coefficients in a table, then scaling the numbers. Filter-design programs are also available.

Once you have obtained the coeffi-

Normally, you design a filter by looking up the appropriate coefficients in a table, then scaling the numbers.

cients you need, you can use the filter several ways. You can calculate the shape of the filter's frequency response and use the shape directly on the Fourier transform of your data signal. You can take the inverse Fourier transform of the filter's shape and use it as a convolution filter. You can calculate the impulse response of the filter and use the resulting numbers as the coefficients of an FIR filter. Finally, you can trans-

form the filter into the Z domain and use the resulting coefficients as an IIR filter. Of course, you can also build some circuit-using capacitors, inductors, and amplifiers to implement your filter, but this article is about digital filters, so we won't discuss that topic.

THE Z TRANSFORM

In classical filter theory, filters are designed in the S domain, where S is the Laplace variable. A particular filter will be represented as a pair of polynomials in the S variable. For example, a 1kHz, sixth-order Butterworth low-pass filter has this representation in the S domain:

1

$$1.625e-23 S^6 + 3.945e-19 S^5 + 4.789e-15 S^4 + 3.685e-11 S^3 + 1.891e-7 S^2 + 6.149e-5 S + 1$$

Digital filters are designed in a different domain—the Z domain. The Z variable represents a time shift by one sample period. Thus, if your data stream is the sequence $D(i)$, then $Z \cdot D(i) = D(i+1)$. Similarly, $D(i)/Z = Z^{-1} \cdot D(i) = D(i-1)$. Z means move forward one position in the sequence, and $1/Z$ means move backwards one position in the sequence. Digital filters work by multiplying filter coefficients by previous values in the data sequence. A digital filter can be considered a polynomial in $1/Z$, with the filter coefficients as the polynomial coefficients.

To design a digital filter, we can use one of three methods. The first method is to draw a picture of what we want and put the picture through an FFT. This method will result in an FIR filter. The second method is to design a classical filter and calculate its impulse response. The resulting numbers are the coefficients of an FIR filter. Finally, we can design a classical filter, then transform the filter polynomials from the S domain to the Z domain. The resulting coefficients will be an IIR filter. The transformation from S domain to Z domain is done with the rational Z transform:

$$S = \frac{2}{T} * \frac{1-Z}{1+Z}$$

Professional Quality

80x86 ROM Development Tools

ROMable DOS, Only \$6 each!

Place your programs in PROM with ROM-DOS. Purchased in quantity, this complete ROMable operating system costs only \$6 per copy. And its small size—less than 32K—helps conserve valuable target system memory. ROM-DOS was designed as an embedded DOS, providing features like power conservation, the ability to run your executables directly in ROM, and full access to built-in devices to support any kind of target system. Plus ROM-DOS provides the power you'd expect from a desktop DOS: a full-featured command processor, extra utilities, batch file support, and more. And now it is compatible with MS-DOS 3.3. ROM-DOS boots and runs from either disk or ROM and can easily be configured to meet your needs. Ask for our free demo disk.



ROM Your MS & Turbo C Code!

Now C_thru_ROM eases ROM development with all versions of Turbo-C, C++, and Microsoft C. New features include support for Turbo Debug and remote debugging via a ROM socket, which does not use a serial port on the target system. C_thru_ROM is a complete ROM development package containing everything you need to work with your choice of compiler and get your application up and running in ROM—quickly. It includes: a full 80x86 locator which outputs in Intel OMF, hex, and binary; ROMable startup code; two choices for remote debugging (a CodeView-like debugger or the Turbo remote debugger); full floating point support; a ROMable library (with *printf*, *malloc*, etc.); and much more. Call, write, or fax for full details.

Call Today Toll-Free 1-800-221-6630

Datalight

17455 68th Avenue NE, Suite 304, Bothell, WA 98011, USA
(206) 486-8086 • fax (206) 486-0253

CIRCLE #240 ON READER SERVICE CARD

AMX

REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 80386
Z80, 64180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
 - task create/delete
 - task priorities
 - memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as
IBM®, Xerox, Hewlett Packard,
Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance
real-time multitasking system
available today.

No Royalties
Source Code Included

Demo Disk and
Manual only \$75 US call for prices for
AMX 86 \$3000 US other processors

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, InSight are trademarks of
KADAK Products Ltd.

KADAK Products Ltd.

206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5



Telephone: (604) 734-2796
Fax: (604) 734-8114

where T is the sample period.

We can transform the Butterworth filter from the S domain to the Z domain by substituting our original formula for S, then cleaning up the polynomials. Here is the result:

$$3.4e-4 + 2.0e-3 Z^{-1} + 5.1e-3 Z^{-2} + 6.8e-3 Z^{-3} + 5.1e-3 Z^{-4} + 2.0e-3 Z^{-5} + 3.4e-4 Z^{-6}$$

$$1 - 3.58 Z^{-1} + 5.66 Z^{-2} - 4.97 Z^{-3} + 2.53 Z^{-4} - 7.05 Z^{-5} + 8.38 Z^{-6}$$

Here is how this formula is used:

$$\text{output sequence} = \frac{\text{input sequence} * \text{filter numerator}}{\text{filter denominator}}$$

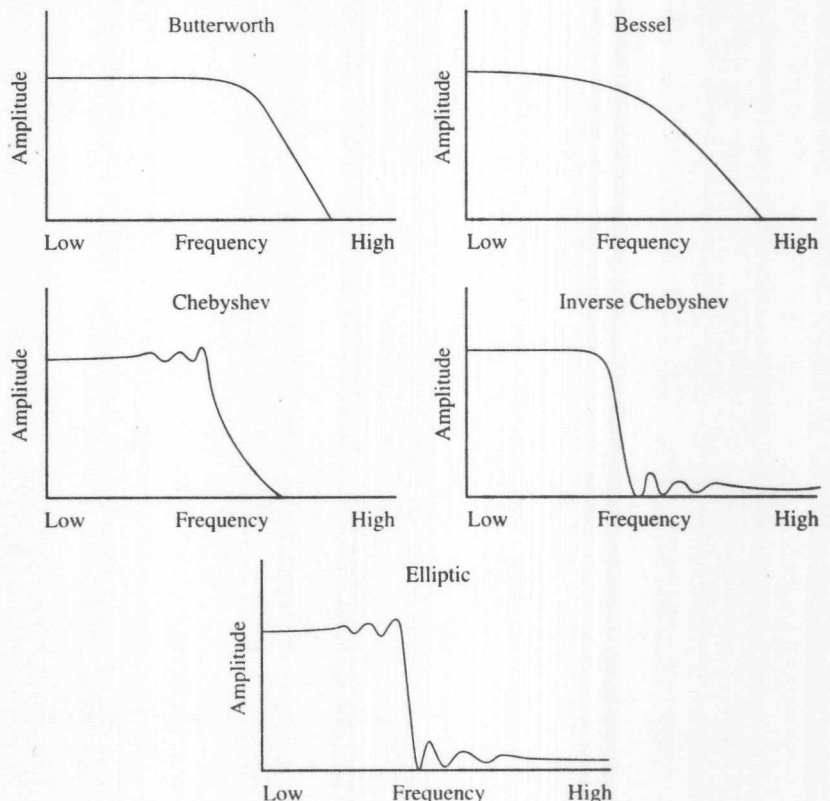
From the formula we just looked at, we see that if the filter polynomial has no denominator, the filter is an FIR because the output sequence depends only

on the input sequence. If the filter has a denominator, the denominator is used with the previous values of the output sequence and the filter is an IIR because the output sequence depends on the input sequence and the previous values of the output sequence.

The order of the Z polynomials tells you how much computation is required to calculate your filter response. Each Z term takes one multiply and one addition operation. For historic reasons, each Z term is called a tap, so an FIR with a 50th order Z polynomial is called a 50-tap filter, and requires 50 multiplies and 50 additions for each output value. Typically, a useful FIR filter will be between the 50th and 200th order, so a lot of arithmetic will be necessary to calculate the filter response. The number of arithmetic operations can be reduced, often by a factor of 1,000 to 10,000, which is why FFT filtering is so popular.

A typical IIR filter will be between

Figure 7
Types of filters.



CIRCLE #242 ON READER SERVICE CARD

Son of DSP Primer

the second and eighth order. These filters are very easy to calculate in comparison to FIR filters. FIR filters can be designed to have linear phase response, making them equivalent to a time delay for phase purposes. This capability is often important in high-speed digital communications, so in spite of their inherent inefficiencies, FIR filters are very popular.

So concludes our whirlwind tour of digital-signal processing. We have explored the basics of DSP math and investigated the famous Fourier transform and how it relates to signal filtering. I hope that now you have a better understanding of how the various pieces of DSP math fit together and

FIR filters can be designed to have linear phase response, making them equivalent to a time delay for phase purposes.

how DSP techniques can be applied in your applications. At the very least, you should be better prepared to work through the more traditional presentations on DSP math and programs.

Mark Lawrence graduated from Cal-

tech with a degree in Electrical Engineering. His company, California Scientific Software, produces *BrainMaker*, a popular neural-network software system. Several CSS products employ DSP algorithms and techniques for filtering, data analysis, and speeding up computations.

REFERENCES:

1. Press, William P., et al. *Numerical Recipes in C*. Cambridge, U.K.: Cambridge University Press, 1988.
2. IEEE DSP Committee. *Programs for Digital Signal Processing*. Washington, D.C.: IEEE Press, 1979.
3. Rabiner, L.R. and B. Gold. *Theory and Applications of Digital Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1975.
4. Brigham, E. Oran. *The Fast Fourier Transform and its Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.
5. Oppenheim, Alan. *Digital Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1975.
6. Lam, Harry. *Analog and Digital Filters*. Englewood Cliffs, N.J.: Prentice-Hall, 1979.
7. Johnson, David. *Introduction to Filter Theory*. Englewood Cliffs, N.J.: Prentice-Hall, 1976.

A STEP BEYOND.

PROMICE takes ROM emulation a step beyond...

An affordable, multi-operational development tool with

ON BOARD INTELLIGENCE

MODULAR DESIGN

SOURCE LEVEL DEBUGGING

FUTURE EXPANDABILITY

The PROMICE enables source level debugging of embedded software by providing communication between the Host and Target systems via the ROM socket. The PROMICE implements ROM monitor based debugging of application code and easily adapts to any target system by simply changing the software required to support the particular target processor.



PROMICE... The Next Generation of Firmware Development Tools

For more information, call or write:

Grammar Engine Inc.

3314 Morse Road
Columbus, Ohio 43231
TEL: 614/471-1113
FAX: 614/475-6871

CIRCLE #244 ON READER SERVICE CARD